Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo

Monografia

Broken Soul
Um jogo para Nintendo DS™

Orientador
Prof. Flávio Soares Correa da Silva

Autores
Marcos Takechi Hirata N.USP: 5123837
Napoleão Nobuyuki Tateoka N.USP: 5489082
Nícia Tiemy Sonoki N.USP: 5382097

Conteúdo

Capítulo 1

Agradecimentos

Ao professor Flávio Soares Correa da Silva pela orientação e interesse pelo nosso projeto e aos amigos que colaboraram com incentivos, ideias e também aqueles que nos ajudaram como *beta testers*.

Capítulo 2

Introdução

Um jogo de computador é um programa de entretenimento virtual onde a plataforma utilizada é o computador pessoal.

Os jogos de computador evoluíram de simples sistemas baseados em texto, com interfaces gráficas simples e de jogabilidade limitada para uma vasta gama de títulos visualmente mais avançada e de jogabilidade infinitamente superior. Muitos deles até permitem que os jogadores interajam entre si, aumentando, assim, a diversão.

2.1 Motivação

Todo jogador de *videogame* um dia já se imaginou criando um jogo com suas próprias idéias, com seus próprios recursos gráficos e sua própria história. Com essa vontade em mente e com uma grande motivação (devido a um Trabalho de Conclusão de Curso de 2005, que também foi um jogo), deu-se inicio à implementação deste projeto.

Jogos são um dos *softwares* mais atrativos e um dos tipos que mais abrangem diferentes áreas da computação, mas que não se limitam apenas a ela. Várias habilidades e conhecimentos são requeridos para a criação de um jogo envolvente, sendo que se destacam a criação de roteiro, habilidades gráficas e conhecimentos em outras áreas, tais como física, história, ciências sociais e cultura geral.

2.2 Objetivos

O objetivo principal do projeto é criar um jogo que utilize as duas telas do Nintendo DS^{TM}e explorar de forma efetiva as funções do *touch screen* em um jogo de *RPG* (*Role-Playing Game*) utilizando todo o embasamento teórico que o curso de Bacharelado em Ciência da Computação nos oferece (principalmente nas áreas de Programação Orientada a Objetos e Engenharia de Software), além de criar uma interface gráfica agradável de se ver.

Capítulo 3

Conceitos e Tecnologias Utilizadas

Neste capítulo são apresentados conceitos necessários para um melhor entendimento deste projeto e mostra quais foram as ferramentas utilizadas durante o desenvolvimento.

3.1 O Jogo

O estilo de jogo escolhido para o projeto foi o *Action RPG*. O *RPG* é um estilo de jogo de interpretação de personagens. O progresso do jogo é determinado por um sistema de regras pré-definido em que o jogador, em seu turno, pode improvisar suas ações e o desenrolar do roteiro é feito gradualmente. O *Action RPG* é um sistema que alia a improvisação, o desenrolar e a ação. A diferença é a participação mais ativa do jogador nas batalhas, ou seja, não é preciso esperar por seu turno para que ele possa fazer seus movimentos, o que garante uma maior dinâmica e interação ao jogo.

3.2 O Ambiente

Para o desenvolvimento do jogo, utilizamos tanto o Microsoft® Windows® como o Linux. A utilização desses dois sistemas operacionais não foi mero capricho, pois muitas das ferramentas que foram utilizadas foram implementadas especificamente para apenas uma das plataformas e, por isso, a utilização dos laboratórios com computadores com o Linux implantado e a utilização de computadores portáteis pessoais com Windows® foram muito convenientes.

As *IDE*s, do inglês *Integraded Development Environment*, utilizadas foram o **VIM** no Linux, o **Visual Studio 2005** e **NotePad++** no Windows®.

3.3 Homebrew

Homebrew é um termo frequentemente aplicado para *videogames* ou outros tipos de *software* produzidos por usuários que visam criar programas para plataformas proprietárias. Um tipo de *software* muito popular são os *fangames*, criados por fãs de desenhos animados, que são jogos de *videogame* ou outro tipo de idolatração que, para o criador do jogo, valha a pena se dar ao trabalho.

3.4 A Linguagem

A linguagem de programação básica utilizada para o desenvolvimento do projeto foi C++. Tal linguagem foi escolhida, pois ofereceu toda a flexibilidade e robustez da linguagem C, além de ter disponibilizado todos os desejáveis recursos de Programação Orientada a Objetos.

3.5 A Biblioteca

Aliado ao C++, a biblioteca escolhida para o desenvolvimento foi o PAlib junto com devkitARM embutido na biblioteca devkitPro.

3.5.1 A PAlib

A *Programmers Arsenal library*, PAlib, é uma biblioteca específica para a criação de *softwares* para o Nintendo DS™e Nintendo DS™Lite. A sua licença é GPL (*General Public License*) e não é uma biblioteca oficiamente apoiada pela Nintendo®.

3.5.2 O devkitPro

devkitPro é um conjunto de ferramentas para o desenvolvimento de jogos caseiros, conhecidos como *homebrew*. Esta biblioteca oferece suporte para o desenvolvimento para vários consoles de *videogames*, tais como PlayStation® Portable(PSP), GameCube™, GameBoy™Advance, Nintendo DS™e Nintendo Wii™.

Dentro desse conjunto de ferramentas existe uma biblioteca específica para cada console, neste caso, o devkitARM, que é para processadores do tipo ARM (Advanced RISC Machine, incluído no DS).

3.6 Nintendo DS™Flash Card

Um Nintendo DS *flash card* é um cartão que substitui um cartucho de jogo original. Ele permite que um usuário possa inserir seus jogos, músicas e vídeos e executá-los no *videogame*. Alguns exemplo de *flash card* são: R4 revolution, Edge, CycloDs e Acekard 2.

Figura 3.1: Flash card

3.7 Nintendo DS™

O Nintendo DS™é um *videogame* portátil criado pela Nintendo® e foi lançado em 2004. Possui duas telas sendo que a inferior é sensível ao toque, possui um microfone embutido e possui suporte à conexão sem fio, além de ser compatível com cartuchos de GameBoy Advance (GBA). Os cartuchos menores de Nintendo DS se encaixam no Slot-1 acima do console, enquanto os jogos de GBA entram no Slot-2, na parte inferior. O Nintendo DS™não é compatível com jogos de Game Boy e do Game Boy Color devido a uma pequena diferença do formato de fábrica e a ausência do processador Zilog Z80 usado nesses sistemas.

Figura 3.2: Nintendo DS™

3.7.1 Acessórios

Apesar da entrada secundária no Nintendo DS aceitar e ter suporte a jogos de Game Boy™Advance, a Nintendo® enfatiza que o principal motivo de ter criado o recurso é para que ele seja a entrada de acessórios que podem ser lançados para o console, portanto a compatibilidade com o GBA seria apenas uma extensão lógica.

O *Rumble Pak* foi o primeiro acessório a utilizar a entrada de expansão. No formato de um cartucho de Game Boy™Advance, o *Rumble Pak* vibra para transmitir as ações em jogos compatíveis, como quando o jogador bate em um obstáculo ou perde pontos de vida. Além desse, muitos outros foram lançados, segue a lista com uma breve descrição:

- Play-Yan: adaptador que permite o Game Boy™Advance SP e Nintendo DS™tocar vídeos MPEG-4 e músicas MP3 a partir de cartões de memória SD;
- Nintendo DS™MP3 Player: tocador de mp3 para o console;
- Nintendo DS™Headset: fone de ouvido e microfone. O microfone é compatível com todos os jogos que o utilizam;
- TV Tuner: receptor de sinais de TV digital;
- Opera web browser: navegador de internet para o portátil;
- Nintendo Wi-Fi USB Connector: podem-se jogar partidas online com o DS mesmo que não haja conexões wi-fi por perto;
- Guitar Grip: acessório criado pela produtora Activision que permite jogar os jogos da série Guitar Hero® no Nintendo DS™. Possui 4 botões.

As letras **DS** foram criadas para significar tanto a expressão *Dual Screen* (duas telas) como *Developers System* (sistema dos desenvolvedores). Esta última se refere às características inovadoras do *gameplay*, ou seja, como o modo de jogar é inovador.

Em meados de março de 2006, a Nintendo® lançou uma versão menor e mais leve do console, o Nintendo DS Lite. Além disso, mudanças de usabilidade foram feitas, tais como o botão de liga-desliga que ficava acima dos botões A, B, X e Y fica agora na lateral direita. O *stylus* ficou maior e está num local de fácil acesso. Segue a especificação simplificada do *hardware* desta versão:

- Duas telas de 3 polegadas sendo que a inferior é sensível a toque;
- Dois processadores: a principal de 67 MHz (ARM946E-S) e um co-processador de 33 MHz ARM7TDMI;
- Resolução: 256x192 pixels;
- Número de cores: 260.000 cores aproximadamente;
- Número máximo de sprites: 128 sprites por cada tela;
- Memória de 4 MB de ram e 64kilobytes de video ram(VRAM) compartilhado com a memória RAM;
- Som estéreo;
- Bateria de lítio com duração entre 6 a 10 horas;
- Capacidade de receber sinais Wi-fi de outros DSś, Nintendo Wiiś e pontos de acesso Wi-fi. Redes WEP encriptadas e não encriptadas são suportadas. Encriptação WPA não suportada;
- Microfone para reconhecimento de voz embutido.

Figura 3.3: Nintendo DS™Lite

Finalmente em outubro de 2008, a Nintendo® apresentou uma nova versão nomeada como Nintendo DSi. Esta possui a mesma característica de sua antecessora, mas possui uma tela 0,25 polegadas maior, uma câmera de 0,3 mega pixels e uma entrada para cartão de memória SD.

Figura 3.4: Nintendo DSi™

3.7.2 Touchscreen

Touchscreen é uma superfície que permite detectar a presença e a localização de um toque. Esse tipo de tela possui dois atributos principais: interação direta na tela e a não necessidade de usar outro dispositivo.

Há três sistemas básicos que são usados para reconhecer um toque na tela:

Resistivo: este sistema consiste em um painel de vidro normal, que é coberto com uma camada metálica resistiva e condutora. Essas duas camadas são separadas por um pequeno espaço e em cima delas, é adicionado uma camada que protege todo esse sistema contra "arranhões". Uma corrente elétrica passa entre essas camadas metálicas e quando o usuário toca a tela, as duas camadas fazem o contato alterando o campo elétrico nesse ponto e as coordenadas são calculadas pelo *software*. Este é o tipo de sistema usado no Nintendo DS^{TM}.

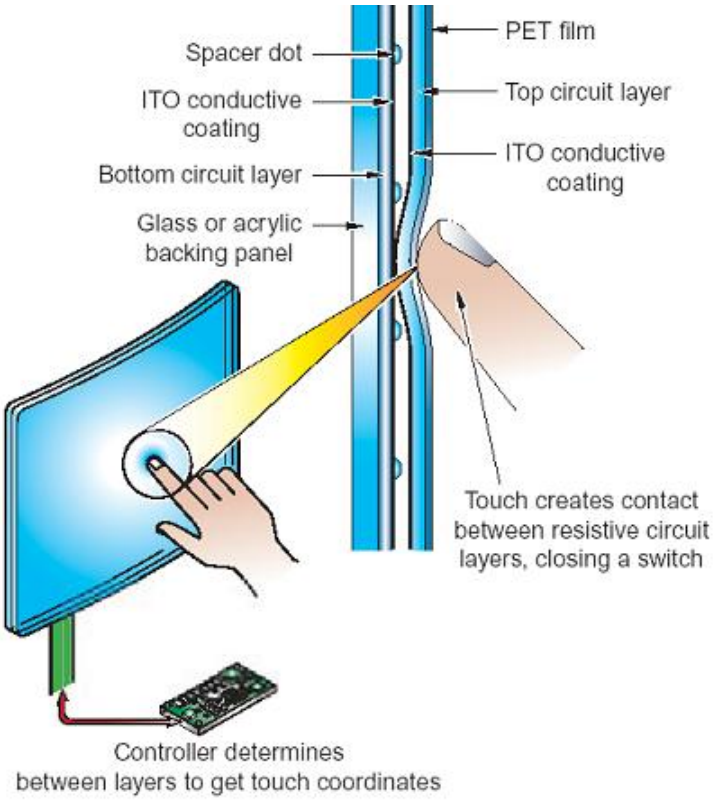

Figura 3.5: Funcionamento do *touch screen* resistivo

Capacitância: neste sistema existe uma camada que armazena carga elétrica que é colocada sobre o painel de vidro. Quando o usuário toca a, uma parte da carga elétrica é transferida para o painel, de forma que a carga na camada capacitiva diminui. Essa diminuição é medida através de circuitos situados em cada canto da tela. O computador calcula a partir das diferenças relativas à carga de cada canto da tela, exatamente onde houve o toque na tela e então envia essas informações para o *software* do *touch screen*.

Ondas Acústicas de Superfície: este sistema usa ondas de ultra-sons que passam sobre o painel tátil. Quando o painel é tocado, uma parte da onda é absorvida. Essa alteração nos registros de onda ultra-sônicos revela a posição do evento de toque e envia essas informações para o controlador para o processamento.

3.8 Ferramentas Utilizadas

Muitas ferramentas auxiliaram o desenvolvimento. As principais e mais relevantes são citadas nessa seção.

3.8.1 Sprites, Cenários e PAGfx

Sprites são pequenas imagens manipuláveis em termos de posicionamento, rotação e distorção que são integradas em apenas uma imagem. Essa imagem é apresentada sempre em dimensão menor que o total de seu tamanho, a fim de mostrar uma pequena parte seguida logo de outra, dando o efeito de animação.

Os *sprites* do Nintendo DS™possuem 3 modos de cores: 16 cores, 256 cores e 16-bit (sprite de uma cor só), sendo que dentre elas foi utilizado apenas o esquema de 256 cores (por motivos estéticos). Esse esquema representa uma paleta de cores que são indexadas a fim de melhorar o rendimento e utilizar menos memória de vídeo, além de diminuir o processamento de cores.

Os *sprites*, no entanto, devem ser utilizados da seguinte forma: cada *tile* (pequenas imagens de tamanho 8x8 *pixels*) deve ser mapeado na memória e indexado para que a paleta de cores possa ser relacionada ao *tile* correspondente. Para isso seria necessária uma grande manipulação de *bits* para criar os *tiles* e então atribuir os dados à memória de vídeo.

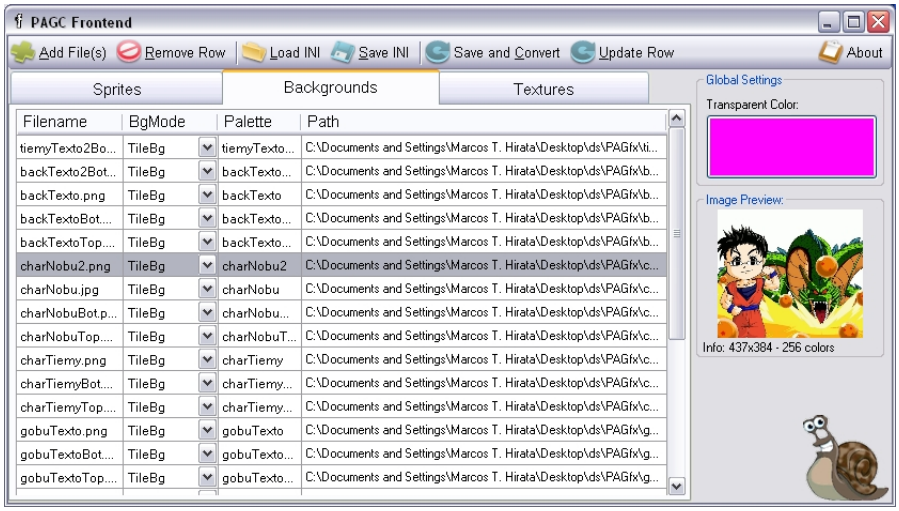

Figura 3.6: Ferramenta PAGfx

A fim de agilizar esse trabalho, a ferramenta **PAGfx** foi utilizada. Este funciona como um conversor de imagens para arquivos em *.c* e *.pal.c*. O primeiro arquivo contém os *tiles* que representam a determinada imagem e o segundo, uma concatenação dos índices dos *tiles* com as cores da paleta. Esses dados são armazenadas em um vetor.

A manipulação para a conversão dos cenários funciona de forma análoga aos *sprites*. A mesma ferramenta é utilizada para a conversão e inclusão das imagens no DS.

3.9 Criação Gráfica

Além de papel, lápis e caneta tinteiro, alguns outros recursos foram utilizados para a criação dos gráficos. A principal ferramenta para edição de imagens foi o *GNU Image Manipulator Program* (GIMP). Esta é uma ferramenta relativamente poderosa e possui bons recursos e é *open-source*. Utilizando o programa, vimos que muitas de suas ferramentas fazem grandes referência à tópicos de matérias que foram vista durante a graduação, tais como B-Splines, Curvas de Bézier e matrizes de cisalhamento (visto em Computação Gráfica, principalmente). O GIMP foi utilizado principalmente para a criação, edição e pintura de sprites e de cenários.

Uma vez editadas as imagens de movimento, foi preciso concatená-las para formarem um *sprite*. Sendo assim, foi criada uma pequena ferramenta, um *script*, que utiliza o comando *montage* para deixá-las no tamanho desejado e rotacioná-las. Após essa manipulação, a concatenação é feita através do *pipe*.

Em relação à criação, 90% dos *sprites* e desenhos foram criados originalmente por integrantes do grupo, sendo que um deles foi o responsável pela criação e os outros membros foram responsáveis pelo auxílio na edição. Alguns cenários foram criados através de *tilesets* do *RPG Maker*, uma ferramenta *freeware* que auxilia usuários fãs de RPG a criarem seu próprio jogo desse mesmo estilo.

3.10 Shell Script

Na linha de comandos de um *shell*, é possível executar diversos comandos para executar tarefas tanto simples, como complexas. Mas diversas vezes essas tarefas são repetitivas, tornando-se fatigantes para o usuário executá-las. Ao juntar uma sequência de comandos em um arquivo de texto simples, obtem um pequeno roteiro do que se deve fazer. O *shell script* vem com o objetivo de automatizar tarefas que são executadas exaustivamente.

Neste projeto, alguns *scripts* foram criados a fim de facilitar e agilizar trabalhos que demorariam horas, sendo que a principal utilização foi para manipular as imagens. O *shell* também foi usado para criar um código com *if*s encadeados. Infelizmente a função da biblioteca utilizada recebe apenas valores constantes, dessa forma o grupo foi forçado a utilizar um método de exaustão para cobrir todos os casos. Renomeação de arquivos também foi uma atividade importante para a organização do projeto, logo um *script* para renomear arquivos também foi implementado.

3.11 Embedded File System

Embedded File System (ou simplesmente EFS) é uma ferramenta bastante útil, que simula um sistema de arquivos virtual em um único arquivo. Este sistema possui um funcionamento bastante interessante, pois faz com que o sistema de arquivos do computador (no coso do projeto, do DS) emule um disco rígido inteiro normalmente e que é armazenado em um único arquivo no *HD* real.

Esse tipo de sistema é muito utilizado em dispositivos portáteis como relógios, *MP3 players*, sistemas controladores de produção numa fábrica, em usinas de energia nuclear, sistemas de monitoramento de sinais vitais, controladores de motores eletroeletrônicos e outros.

Uma das dificuldades ao longo do projeto foi o manuseio da memória RAM. Como citado anteriormente, a memória de vídeo é compartilhada com ela, logo, não era possível carregar um arquivo mp3 inteiro na memória, pois não haveria recursos para outros processos.

Com o *EFS*, foi possível carregar os cenários à medida que eram necessários e após usá-los, eram descarregados. Já os arquivos de músicas foram executados por *stream*, ou seja, pedaços do arquivo eram carregados na memória de acordo com a execução do *.mp3*.

3.12 Emulador

Um emulador é um *software* que expõe as funções de um sistema para reproduzir seu comportamento, permitindo que um *software* criado para uma plataforma funcione em outra. Ele também é responsável pela simulação, virtualização dos circuitos integrados ou dos *chips* do sistema de hardware em um *software*.

Uma das vantagens de se utilizar um emulador é poder testar um *software* ainda em desenvolvimento sem se preocupar se ele danificará o sistema hospedeiro. Mas em contrapartida, a velocidade de execução pode ser muito diferente da real e, além disso, pode não simular fielmente o *hardware* ou um sistema alvo. Esse tipo de *software* é muito utilizado em ambiente de desenvolvimento e em *videogames*. Alguns exemplos de emuladores: Virtual Box, VMware®, No$gba, iDeas e ZSNES (eses três últimos, emuladores de *videogame*).

Para esse projeto, foi utilizado o No$gba. Este é um emulador gratuito e não oficial criado para emular o console Nintendo DS™e GBA™.

Capítulo 4

A Implementação

Neste capítulo serão descritas as principais etapas da implementação do jogo, as dificuldades e soluções encontradas para cada problema.

O projeto iniciou-se com o levantamento de requisitos e planejamento, que serão melhor detalhados a seguir.

4.1 Planejamento

O Planejamento foi a primeira atividade a ser executada a fim de fornecer uma direção para a equipe com base no escopo definido para o projeto e nas datas de entrega.

Dadas as dificuldades encontradas, principalmente com a falta de documentação e falhas de implementação das bibliotecas utilizadas, o plano foi constantemente modificado para reacomodar as atividades. Além disso, existiram muitas mudanças no escopo devido aos *feedbacks* do orientador e de outros colaboradores.

O cronograma abaixo mostra o plano de "granularidade grossa". Este foi montado baseado nos módulos a serem desevolvidos.

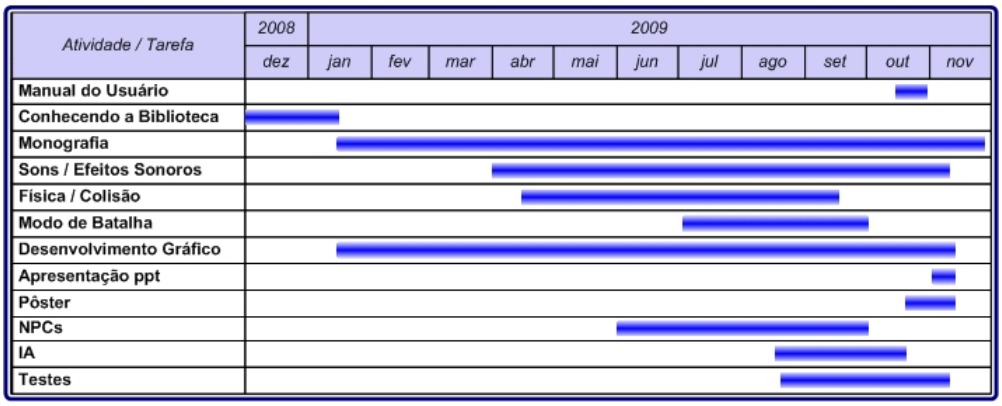

Figura 4.1: Cronograma do projeto

4.2 Método de Trabalho

O método de trabalho utilizado ao longo do projeto foi calcado em métodos ágeis. Tal método possui algumas características que, para um grupo pequeno com alto nível de produtividade, seria imprescindível seguir. Seguem os motivos relacionados ao quesito relações interpessoais:

- Respeito e confiança mútua;

- Autonomia e autoridade para tomada de decisões: a equipe deve ter autonomia e autoridade de forma descentralizada a fim de garantir liberdade de resolução de problemas, com cada um visando uma solução em relação a um problema ou objetivo técnico do projeto;
- Desenvolvedores, clientes e gestores devem atuar colaborativamente. Nesse caso, os beta *testers* fizeram o papel de cliente fornecendo informações e conselhos.

Em relação ao quesito técnicas de desenvolvimento, os principais métodos adotados foram:

- Modelo de desenvolvimento incremental (muitas vezes com desenvolvimento pareado);
- Entrega de software útil com alta frequência, ou seja, funcionalidades ou módulos entregues num pequeno intervalo de tempo;
- Progresso do projeto monitorado a partir dos incrementos entregue pelo grupo;
- Discussões face-a-face privilegiadas, tanto entre desenvolvedores como entre desenvolvedores e clientes;
- Modificações nos requisitos esperados, ou seja, mudanças no desenvolvimento ou nos objetivos do projeto não são esperados como se fossem algo ruim, mas sim bem vindas, pois atualizam e provavelmente melhoram alguma funcionalidade do sistema;
- Simplicidade do código, fazendo com que as chamadas das funções falem por si mesmas;

4.2.1 O Modelo Incremental

O modelo incremental foi o melhor modelo aplicado ao projeto. Cada desenvolvedor foi responsável pela produção de "incrementos", que são estratificações do *software* como um todo. O modelo aplica uma sequência linear de uma forma racional à medida que o tempo passa, sendo que cada sequência produz incrementos do software passíveis de serem entregues. Quando o primeiro incremento é entregue, geralmente ele é chamado de *núcleo do produto*.

Os requisitos básicos do produto foram entregues e a partir do núcleo do produto as funcionalidades complementares ou suplementares foram adicionadas. Um plano foi desenvolvido para os próximos incrementos e, nesse caso, foi criado um cronograma de atividades a fim de impôr um limite de tempo e uma lista de tarefas a serem cumpridas. O desenvolvimento incremental é muito útil no caso de sistemas a serem desenvolvidos com pouca mão de obra dentro de um prazo de entrega curto, sendo este exatamente o caso deste projeto.

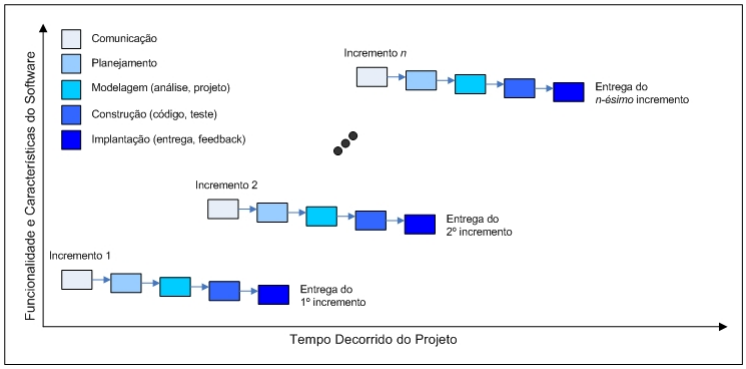

Figura 4.2: Esquema do modelo incremental

4.3 Requisitos Funcionais

Os requisitos funcionais estão intimamente ligados às funcionalidades propostas inicialmente pelo sistema, ou seja, descrevem o que o sistema deve fazer e como o sistema deve reagir para as entradas esperadas e não esperadas. Após algumas reuniões, foi decidido que o projeto deveria atender às seguintes características:

1. Usar o *touch screen* para dar golpes do personagem principal: ao fazer uma reta, lançar uma magia na direção desejada e ao desenhar uma circunferência, desferir o golpe giratório;
2. Conter pelo menos uma *quest* (objetivo a ser cumprido);
3. O jogo seria um *action rpg*;
4. Ter uma abertura contando a história do jogo;
5. Ao andar pelo mapa, o herói encontraria uma batalha aleatóriamente;
6. Criar um demo jogável;
7. Poder caminhar de um cenário para o outro;
8. Ter um menu de entrada no jogo;
9. Quando o inimigo morresse, o herói ganharia alguns pontos de experiência;
10. Poder trocar de espírito durante a batalha;
11. Poder carregar e usar alguns itens;
12. Conversar com os *NPC*s;
13. Tocar efeitos sonoros e músicas de fundo;
14. Cnter uma interface condizente com a qualidade gráfica que o console pode proporcionar.

4.4 Levantamento de Requisitos de Usabilidade

Alguns rascunhos do produto final foram criados a fim de realizar testes de usabilidade com cinco pessoas que se voluntariaram a serem avaliadores do projeto. Todos os cinco são alunos do BCC e seus nomes foram citados como colabores do projeto nos créditos do jogo. Foram avaliados principalmente 4 itens:

1. Relação entre sistema e mundo real;
2. *Design* estético e minimalista;
3. Consistência e padronização;
4. Reconhecimento e não recordação;

Sendo o projeto baseado numa interface de manipulação direta, foi fácil introduzir o conceito do jogo aos avaliadores (os quais também já haviam tido contato com sistemas semelhantes) e assim receber um parecer sobre os itens relacionados acima.

A relação entre o sistema e o mundo real: foram avaliados o tipo de linguagem empregada no texto e a consistência com sistemas semelhantes em relação ao que estava sendo desenvolvido.

Design estético e minimalista: avaliaram a clareza e a organização da interface, verificando a presenạ de informações irrelevantes.

Consistência e padronização: foi mantido um padrão a fim de guiar o usuário aos próximos passos do sistema de forma intuitiva.

Reconhecimento e não recordação: o reconhecimento ao invés da recordação é aconselhado a fim de facilitar o uso do sistema pelo usuário. Ações e opções visíveis e acessíveis são altamente recomendadas.

Algumas modificações na interface foram sugeridas em relação ao *design* inicial, tais como agrupamentos diferentes de atributos do herói no menu de *status* e exclusão de botões virtuais para o manuseio do menu superior (tela que não possui a propriedade do *touch*). Como uma das principais motivações do projeto era utilizar o *touchscreen* efetivamente, foi projetado o manuseio do menu superior através da aparição de botões virtuais na tela inferior de modo que o *stylus* pudesse ser usado durante o maior tempo possível. Mas a implementação dessa funcionalidade foi abortada, pois tornaria o direcional inútil fora de batalha e também quebraria o padrão para usuários já acostumados com jogos nesse estilo para o Nintendo DS™.

4.4.1 CRAP

Sigla utilizada para Contraste, Repetição, Alinhamento e Proximidade, Ã© uma técnica de IHC criada por Greenberg, professor do departamento de computação da Universidade de Calgary, Canadá, que foi utilizada em vários momentos a fim de criar uma interface limpa e clara de modo fácil e sem muitas complicações. Essas pequenas regras foram tidas como conceitos básicos de uma boa interface para o grupo.

Contraste : realce dos elementos dominantes, fazendo com que coisas diferentes pareçam diferentes.

Repetição : manter a consistência e repetir o *design* ao longo do projeto para que uma unidade visual seja criada.

Alinhamento : conexão visual dos elementos a fim de criar um fluxo visual do conjunto de informações apresentado.

Proximidade : agrupamento de elementos que possui algum tipo de relação seja por atributo, relevância, cor ou qualquer outro tipo de identidade.

O menu que acompanha o jogo, bem como os menus de batalha e de status do heói foi criado a partir desses conceitos.

4.5 A Arquitetura

Como o jogo criado é um demo, a arquitetura do software foi baseada em uma máquina de estados que mostra o funcionamento do jogo e na versão atual, pode-se notar que não há um estado final, ou seja, o jogo não acaba. Este esquema de arquitetura foi criado para que pudesse ser relativamente pequeno, inteligível e que mostrasse como os componentes se relacionam entre si no sistema.

Cada (conjunto de) módulo(s) ou função(ões) de grande importância do jogo é (são) representada(s) por estados que são acessados através de chamadas não nulas que retornam algum tipo de resposta ao controlador do sistema. O núcleo é o responsável pelo controle, distribuição de tarefas e organização das *quests*. Ele inclui os módulos responsáveis pela navegação do personagem pelo mundo, a instanciação do herói e o direcionamento do texto ao jogador nos momentos apropriados do jogo, além de escolher as músicas e efeitos sonoros para cada estado do jogo.

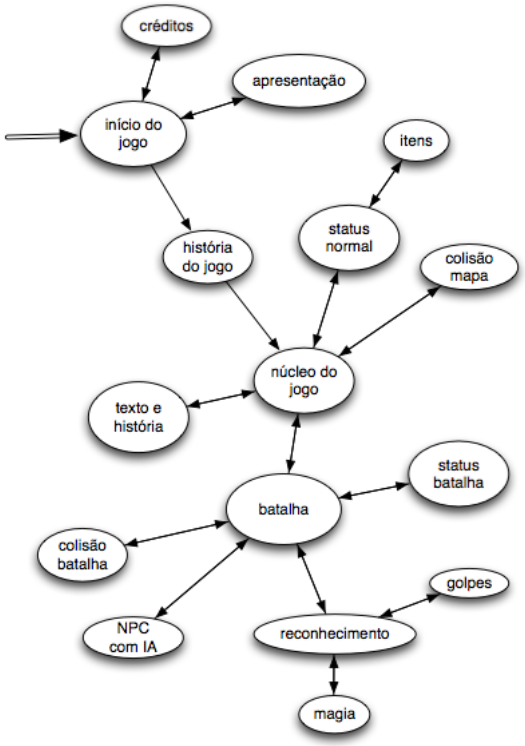

Figura 4.3: Máquina de estados do jogo

4.6 Usuários

O projeto foi concebido através da participação ativa de usuários assíduos de jogos, consultando-os sobre suas experiências e conhecimentos sobre produtos semelhantes ao que estava sendo desenvolvido. Dessa forma, a criação da interação pode ser acompanhada não só pela experiência do desenvolvedor e de pesquisas previamente realizadas, mas também através de opiniões diversas (as quais eventualmente podiam divergir) que foram importantes para a ponderação na intervenção ou andamento da criação.

Um pedido recorrente pelos usuários foi um *feedback* a cada evento ocorrido no jogo. Um exemplo de aplicação do *feedback* é um movimento de recuo que acontece quando o herói recebe algum dano do inimigo ou vice-versa, pois o decréscimo do valor na tela superior do status não era chamativo o suficiente pra indicar ao usuário que o inimigo ou o herói haviam recebido dano.

Em relação aos comandos de ação durante a batalha, muitos foram criados a partir de movimentos que já existiam a fim de facilitar o aprendizado.

Um manual do usuário foi criado para o auxílio dos jogadores que nunca haviam tido contato com tal plataforma ou estilo de jogo. Muitas figuras com instruções embutidas foram adicionadas a fim de facilitar e ilustrar de maneira efetiva o trabalho de aprendizagem do usuário.

4.7 Batalha

Enquanto o herói anda pelo mapa ou pelo cenário simplesmente por vontade de conhecê-lo ou a fim de cumprir algum objetivo, uma função aleatória entra em ação. Esta é responsável pelo encontro dos inimigos com o herói, iniciando o módulo de batalha. O cenário de batalha é um dos módulos mais importantes do jogo, pois é o que dá ação e dinamicidade ao jogo. Foi também o responsável por boa parte da demanda do trabalho de programação. O módulo de batalha engloba funções de inteligência artificial, reconhecimento de formas, conceitos em Interação Humano-Computador e POO.

4.7.1 Iniciando a Batalha

Ao confirmar o início da batalha, uma função embaralha a imagem da tela a fim de dar um efeito de transição da tela do mapa para o fundo da batalha. Os cenários e posições, bem como objetos e NPCs presentes antes do início da batalha têm seus dados armazenados (para que, ao fim da luta, tudo possa voltar ao seu devido lugar novamente).

Na tela superior, um novo *background* de status é carregado, mostrando os atributos e fotos dos inimigos. Em relação ao herói, a tela de status mostra seus pontos de vida, pontos de magia e o espírito utilizado por ele. Já na tela inferior, *sprites* de inimigos e uma nova sequência de **sprites** do herói são carregados, um cenário de batalha relacionado ao local onde o herói estava no mapa aparece e novos comandos de ação são carregados para que o herói possa interagir contra os inimigos.

4.7.2 Reconhecendo Movimentos

O Nintendo DS™possui diversos jogos que utilizam inúmeras funções de reconhecimento de formas e letras, um bom exemplo de jogo para reconhecimento de letras é *Kanji Sono Mama*™, um software que reconhece ideogramas japoneses e d'a o seu significado como se fosse um dicion'ario japonês-inglês. Para o projeto, foram implementadas funções simples de reconhecimento de formas geométricas simples e comandos simples de ações comuns.

Figura 4.4: Duplo toque para dar a espadada

O jogador pode atacar o inimigo com um golpe normal dando um duplo toque na tela. O principal fator para este reconhecimento é o tempo que o usuário deixa o *stylus* pressionado na tela.

Figura 4.5: Desenho da direção para lançar magia

Para iniciar o processo de invocação da bola de fogo, o jogador deve segurar o direcional para baixo e desenhar um risco com início em algum ponto do sprite do herói em direção ao local onde ele quer lançar a magia. No momento que o jogador deixa de pressionar a tela, o reconhecimento é terminado. Com o ponto inicial no herói e o ponto final no local onde o *stylus* deixou de pressionar a tela, é possível calcular o ângulo em que a magia percorrerá a tela em relação a uma paralela ao eixo X.

Para desferir o golpe rotatório, o jogador deve desenhar um círculo. Esse algoritmo merece uma seção à parte para a explicação de sua implementação.

4.7.3 Reconhecendo a Circunferência

Para um divertimento maior do jogador e também para uma utilização um pouco mais avançada do *stylus*, a implementação do reconhecimento da circunferência foi proposta pelo grupo. Como um sistema de reconhecimento de formas genérico, que reconhece circunferências de todas as formas, em si acarreta em um trabalho de implementação e pesquisa muito grande, foi decidido fazer uma pequena pesquisa com a população do IME para poder provar que a maioria das pessoas desenham de um determinado modo.

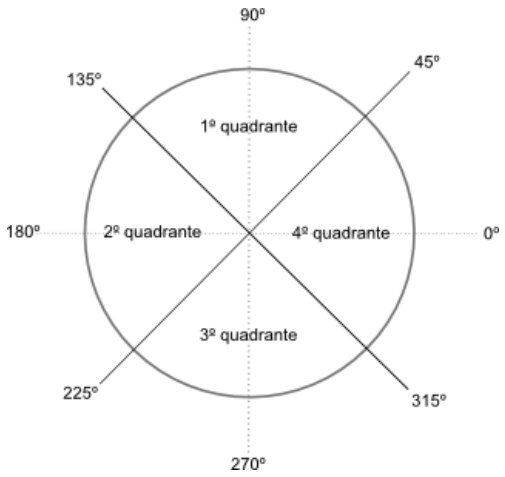

Figura 4.6: Modelo conceitual da circunferência

Foram coletadas 300 amostras de desenhos de circunferências feitas por alunos da graduação de cursos variados do IME-USP (Licenciatura em Matemática, Bacharelado em Matemática, Bacharelado em Matemática Aplicada Computacional, Bacharelado em Matemática Aplicada, Bacharelado em Estatística e Bacharelado em Ciências da Computação) e foi medido o número de ciclos do processador do console Nintendo DS^{TM}que cada pessoa demora para desenhá-las. Cada participante da pesquisa foi orientado a desenhar uma circunferência de forma rápida, como se fosse circular uma resposta numa prova de múltipla escolha, tendo certeza de que ela estaria certa.

A tabela mostra os resultados da pesquisa. As linhas da tabela são as variáveis analisadas no desenho:

- 1° quadrante: número de pessoas que começaram a desenhar a circunferência no primeiro quadrante (conforme definido na Figura 4.6);
- outros quadrantes: número de pessoas que começaram a desenhar a circunferência no segundo, terceiro ou quarto quadrante (conforme definido na Figura 4.6);
- sentido anti-horário: número de pessoas que desenham a circunferência no sentido anti-horário;
- sentido horário: número de pessoas que desenham a circunferência no sentido horário;
- fecha: número de pessoas que fecham a circunferência;
- não fecha: número de pessoas que não fecham a circunferência.

Variável analisada	Quantidade de pessoas
1º quadrante	197
outros quadrantes	103
sentido anti-horário	267
sentido horário	33
fecha	246
não fecha	54

Figura 4.7: Resultados da pesquisa

Também foi feita uma análise do número de ciclos que as pessoas demoraram para desenhar.

Número de ciclos	Quantidade de pessoas
13	1
14	4
15	5
16	9
17	5
18	3
19	10
20	9
21	3
22	3
23	4
24	7
25	3
27	3
28	3
29	1
30	1
31	1
32	2
33	2
34	1
44	1

Figura 4.8: Resultados dos ciclos

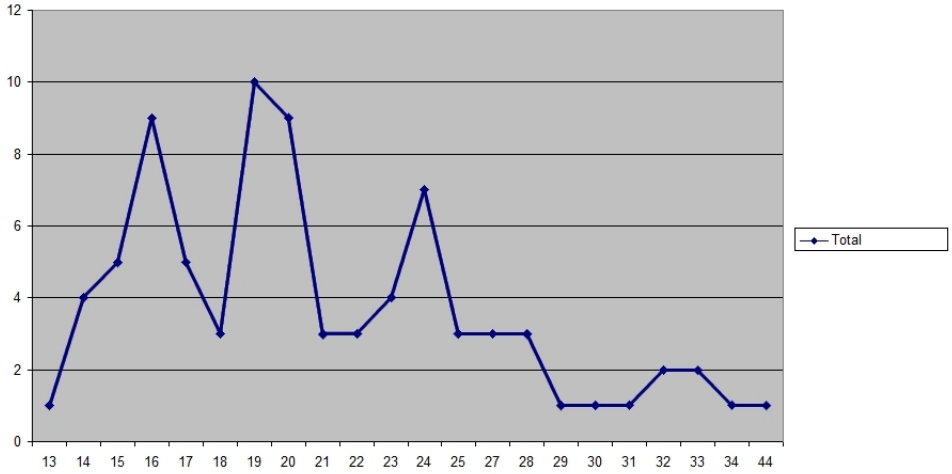

Figura 4.9: Gráfico com número de ciclos

Através destes resultados, pode-se calcular que a média dos tempos de desenho é 21,3 ciclos e a variância é 5,7. Com isso, concluiu-se que a maioria das pessoas levam entre 15,6 e 27 ciclos para desenhar uma circunferência. Deste modo, ficou definido no código que o intervalo a ser reconhecido é de 15 a 27 ciclos.

O algoritmo reconhece a composição do movimento do *stylus* no eixo X e no eixo Y. No eixo X, ele verifica se o usuário fez o desenho da direita para a esquerda, da esquerda para a direita e mais uma vez, da direita para a esquerda. No eixo Y, ele verifica se o movimento foi de cima para baixo e, logo em seguida, de baixo para cima.

Figura 4.10: Desenho da circunferência para o golpe giratório

O tempo vem a garantir que o sistema não reconheça circunferências muito pequenas (pessoas que demoram menos que X ciclos) e que pessoas que queiram utilizar o desenho da circunferência para andarem na tela (que demoram mais de Y ciclos) não desfiram o golpe circular, ou seja, se o tempo ultrapassar os limites previstos ou o *stylus* demorar menos tempo que o estabelecido, o golpe não é executado.

4.7.4 Sistemática de Manuseio

O Nintendo DS^{TM}foi projetado a fim de oferecer uma maior interação entre jogador e o sistema através do *stylus* e do *dual screen*. Isso demanda um estudo de usabilidade um pouco mais cuidadoso a fim de fazer com que os jogadores possam desfrutar do jogo sem perda de foco ou sequência, ou seja, para que o jogador não precise ficar desviando o olhar da tela desnecessariamente ou para que as várias funcionalidades que o Nintendo DS^{TM}oferece não sejam usadas de forma confusa e atrapalhem a diversão do usuário.

O jogo foi projetado para que a pessoa utilize o *stylus* com a mão direita e fique com a mão esquerda segurando o console. O herói pode ser totalmente controlado apenas com o *stylus*, mas para uma maior interatividade e diversão foram adicionados comandos com a mão esquerda. Apesar de utilizar a mão esquerda parecer um complicador, a simplicidade demasiada em jogos de ação foram consideradas sem graça pelos *beta-testers*, por isso comandos adicionais foram implementados a fim de dar uma maior dinamicidade e interatividade ao jogador. Pensando na posição dos dedos ao segurar o console foi decidido que:

- Pressionar o direcional para baixo com o dedão esquerdo: aciona o reconhecimento do desenho da direção que se deseja projetar a magia (ao desenhá-la com o *stylus*);
- Pressionar o botão L com o dedo indicador esquerdo: incorpora a alma que garante habilidades diferentes;

Figura 4.11: Usuário segurando o DS

Pressionar o direcional para baixo com o dedão esquerdo foi definido dessa forma devido à posição naturalmente próxima dele ao segurar o console, assim como o dedo indicador esquerdo fica próximo ao botão L.

Nessa configuração, é preciso apenas uma movimentação mínima de dedos para que novos eventos possam ocorrer. Dentre os cinco avaliadores, quatro consentiram, após um treinamento, que a configuração é de fácil uso e memorização.

Em relação ao uso do *stylus*, os comandos foram baseados em movimentos de uso simples que usuários de computadores domésticos estão acostumados a utilizar.

- Pressão simples: leva o herói a seguir o ponto tocado;
- Duplo toque: ataque simples;

A pressão simples é baseada no *drag-and-drop* de sistemas operacionais gráficos. Basta o usuário "arrastar"o herói até o ponto desejado. O duplo toque, também baseado num dos principais movimentos do mouse, indica que alguma ação mais significativa foi tomada, logo o golpe da espadada foi atribuído a ele.

O último movimento implementado foi o desenho da circunferência. Este movimento intuitivamente remete ao jogador o golpe giratório.

4.7.5 Inteligência Artificial

Em jogos de RPG, a inteligência artificial é um recurso extremamente essencial ao divertimento do jogador, pois são funções implementadas a fim de dar a impressão de vida própria aos outros personagens presentes no jogo (tais como movimentação própria, ações que correspondem ao estado atual do jogo, etc). Nesta seção, uma breve explicação em relação a IA dos inimigos será feita.

Durante a batalha, alguns inimigos tem o objetivo de atacar o jogador, mas outros podem apenas ficar fugindo a fim de irritar o usuário. Quando os inimigos encontram-se em situação perigosa, no caso, com um quarto de seus pontos de vida, eles mostram sua "personalidade verdadeira". Alguns arquétipos foram utilizados para definir cada personalidade:

- Medroso: o inimigo foge em direção contrária à posição do herói;
- Agressivo: o inimigo aumenta sua velocidade de movimento e corre em direção ao heroi para atacá-lo com mais força;
- Traiçoeiro: o inimigo finge que está com medo do herói e foge dele, mas quando o jogador chega perto para atacá-lo, o monstro repentinamente muda sua direção e ataca o jogador com uma velocidade de movimento maior a fim de obter mais chance de acertar o usuário. Se o herói começar a fugir, o inimigo se afasta dele a fim de incitar mais uma perseguição;

4.7.6 Magia

As magias são ataques que utilizam pontos mágicos do herói (MP - *Magic Points*). Elas dão uma emoção a mais ao jogo, dado que são golpes especiais adquiridos no desenrolar do jogo e elas podem evoluir durante a história, tornando-se mais fortes. Existem muitas magias com propriedades diferentes, tais como recuperação de dano, ataque (retirando pontos de vida do inimigo) ou atribuição de efeitos especiais ao jogador, como alta velocidade, super força ou qualquer outro efeito que possa ser considerado positivo para o herói. No caso do Broken Soul, a magia implementada surte efeito de ataque ao inimigo.

Para desferir a magia, foi criada uma *flag* que identifica se o herói está utilizando o espírito ou não.

Para a implementação da esfera de fogo foi criada uma classe Magia.cpp que implementa um projétil. O herói poderá desferir a magia com o movimento esperado pelo reconhecedor, o projétil percorrerá a tela e desaparecerá após um determinado tempo ou após colidir com um inimigo. Ao colidir, o projétil transforma-se em fogo e paralisa o personagem hostil durante um certo tempo.

Para a implementação levou-se em conta a facilidade de direcionar a magia ao inimigo. Para isso, um algoritmo de reconhecimento de reta foi implementado para que o jogador pudesse utilizar o *stylus* e lançar sua magia em qualquer direção da maneira mais fácil possível.

4.8 Menu/Status

O Menu/Status do jogo é fonte de informações essenciais aos jogador de RPG. Com elas o jogador pode montar estratégias a partir de seu estado atual, além de poder acessar seu inventário de itens para utilizar algum objeto que surta efeitos especiais em seus atributos.

Foi decidido deixar uma tela apenas para mostrar o status do herói (no caso, a tela superior) para que a tela inferior pudesse comportar toda a parte gráfica animada do jogo e para que o jogador pudesse focar sua diversão exclusivamente nela, tendo mais espaço para utilizar o stylus em uma área maior (a máxima possível, no caso). A separação em *submenus* visa agrupar atributos de características semelhantes a fim de facilitar a consulta do jogador ao status do herói ou aos itens que ele carrega.

Quando o jogador está fora de uma batalha, ele pode verificar os seus atributos (pontos de vida, pontos de magia, pontos de experiência, etc), verificar em qual nível está e abrir *submenus* para utilizar itens ou verificar quais espíritos o herói possui.

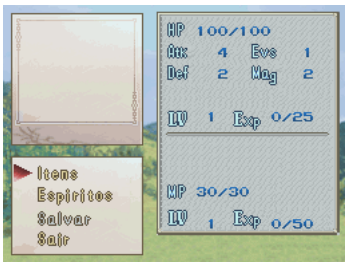

Figura 4.12: *Status* do herói fora da batalha

Os itens são objetos acumulativos que são armazenados em uma lista ligada com as operações básicas (inserir, remover, procurar). Os itens têm efeitos diferentes no herói, mas no atual estado do projeto foi implementada apenas a recuperação de pontos de vida.

Figura 4.13: Menu de itens.

Ao entrar em uma batalha, as informações são reorganizadas em outra interface com dados minimalistas (apenas HP, MP e nome do espírito utilizado) em relação ao status do herói para que os atributos e fotos dos inimigos também possam aparecer, informando ao jogador alguns detalhes sobre seus oponentes.

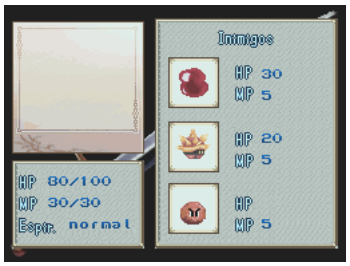

Figura 4.14: Tela do menu na batalha

4.9 NPC (*Non-Playable Character*)

Os NPCs são personagens que não estão sob o controle do jogador. Geralmente têm a conotação de que sempre são aliados ou ao menos personagens neutros no jogo, enquanto os personagens *hostis* são chamados de inimigos, *creeps* ou *mobs*. O comportamento dos NPCs é usualmente definido pelo programador ou certas ações são disparadas por diálogos entre eles e o jogador.

No projeto, os NPCs foram espalhados por todos os cenários do mundo. Existe pelo menos um NPC em cada cenário que é responsável pelo fornecimento de informações sobre o local, tal como a identificação do cenário (o nome) e a situação atual de cada lugar.

No cenário da cidade, vários NPCs estão agrupados a fim de fornecer informações (como dicas e objetivos) sobre a *quest* atual. As informações são obtidas através de um banco de dados rudimentar criado para guardar as falas de cada NPC. A cada objetivo, as falas dos NPCs são trocadas a fim de fornecer novas informações sobre a próxima *quest* a ser cumprida, logo, no banco de dados, os dizeres de cada um são indexados pelo ID do NPC e pela identificação da *quest* atual.

Uma inteligência artificial de movimentação simples foi implementada para os personagens não jogáveis. Eles apenas andam em linha reta em um ciclo vai-e-volta, ou na horizontal ou na vertical. A cada personagem é possível atribuir a distância que ele percorrerá e o tempo que ficará parado depois que percorrer o trajeto. Há também a opção de o NPC ficar parado.

Figura 4.15: Exemplos de NPCs

4.10 Banco de Dados

Um banco de dados é uma coleção de dados inter-relacionados, representando informações sobre um domínio específico, por exemplo, uma lista telefônica.

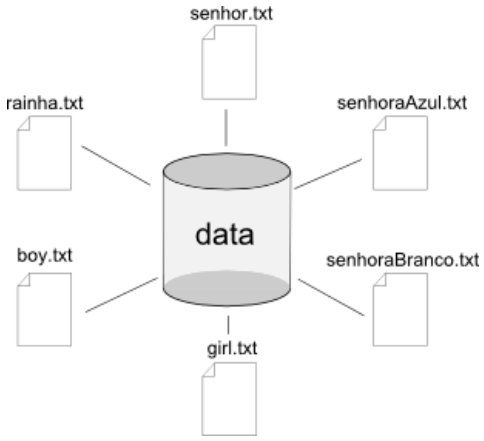

Figura 4.16: Modelo de banco de dados

Como as falas dos personagens podem ser longas, guardá-las na memória seria inviável e com esse intuito a equipe desenvolveu um banco de dados primitivo. As tabelas desse banco são representadas como arquivos textos e em cada arquivo são guardados os dizeres de cada personagem.

A estrutura destas tabelas é bem simples, como mostra o trecho abaixo:

```
2
111Os monstros da floresta estao cada vez mais selvagens. Cuidado! Voce esta aqui fazendo
um favor pro seu amigo,
30mas corre alto risco de vida.
1
106Gracas a voce, o senhor da sua cidade pode visitar o tumula de sua esposa. Ele vai ser
eternamente grato.
```

A primeira linha indica o número de frases que o NPC possui, enquanto na segunda linha, um outro número precede a fala do NPC. Esse número identifica quantos caracteres existem nessa frase do NPC. Essa informação é muito relevantes devido à limitação da biblioteca utilizada. Ela suporta no máximo frases com 192 caracteres. Até seria possível omitir essa informação e contar os caracteres em tempo real, mas devido ao baixo poder de processamento do console, isto prejudicaria o desempenho do jogo.

Essas duas informações constituem um bloco que, por sua vez, representa os dizeres da quest. Logo, todos os personagens possuem a mesma quantidade de sequência de blocos.

4.11 Quests

Quests são missões em que o jogador resolve enigmas ou quebra-cabeças para receber uma recompensa ou continuar com o desenvolvimento da história. Elas possuem uma importância significativa durante o jogo, pois fazem com o que jogador fique entretido com o enredo. São o principal fator responsável pelo envolvimento do usuário em um jogo de *RPG*.

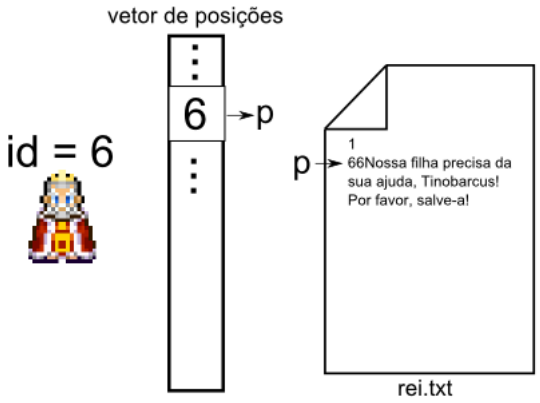

Figura 4.17: Funcionamento das *quests*

A cada *NPC* é atribuído um índice. Entre outras utilizades, este índice é usado como chave primária para o banco de dados de falas. Cada *NPC* possui um ponteiro de leitura do arquivo que é armazanado num vetor e a posição dele é o indentificador do personagem.

Assim, quando o jogador toca no *NPC*, a leitura da próxima fala é feita diretamente sem percorrer todo o arquivo. Ao completar a *quest*, as falas dessa missão são ignoradas e o ponteiro da posição do arquivo para a próxima fala é guardado no vetor.

4.12 Colisão

Em simulações físicas, *videogames* e geometria computacional, algoritmos de detecção de colisões são os responsáveis pela simulação da realidade nos sistemas virtuais criados como ambiente de teste. Em vários casos, a detecção mal feita pode acarretar respostas errôneas, o que leva à falhas na simulação.

Existem dois tipos de detecção de colisão, detecção à *priori* e detecção à *posteriori*. A detecção à *priori* verifica se o próximo passo pode ser dado, ou seja, se na próxima iteração o objeto colide com algo. Se ocorrer a colisão no próximo passo, o algoritmo impossibilita esse movimento. A detecção à *posteriori* move o objeto e depois verifica se ocorreu a colisão. Se ocorreu a colisão, o objeto é movido para o passo anterior.

O Broken Soul utiliza a detecção à priori em suas simulações. A seguir o processo de cálculo de colisões é descrito.

4.12.1 Colisão com o Cenário

Todos os cenários, *backgrounds*, possuem um *Map*. Um *Map* é um mapeamento do *background* em *tiles* de 8 por 8 *pixels*. Os *tiles* são indexados começando pelo primeiro *tile* superior esquerdo, que indica o *tile* transparente e possui índice zero. Os *tiles* com cores diferentes da transparente à direita dele têm seus índices incrementados em um.

Figura 4.18: Valores dos *tiles*

O mapa de colisão é convertido para um *Map* através da ferramenta **PAGfx** que gera dois arquivos: *.c* e *.pal.c*. O segundo é o mapeamento de cores que, para este fim, não será necessário. O Map está contido no primeiro arquivo e é representado por um vetor. Em cada posição está contido o índice do *tile* mapeado.

O motivo pelo qual o *Map* é guardado num vetor é para armazenar de forma contínua as posições dele na memória RAM do console. Se uma matriz fosse utilizada para esse fim, possivelmente as suas linhas seriam guardadas em posições distantes o que aumentaria o tempo de acesso para uma determinada posição dela.

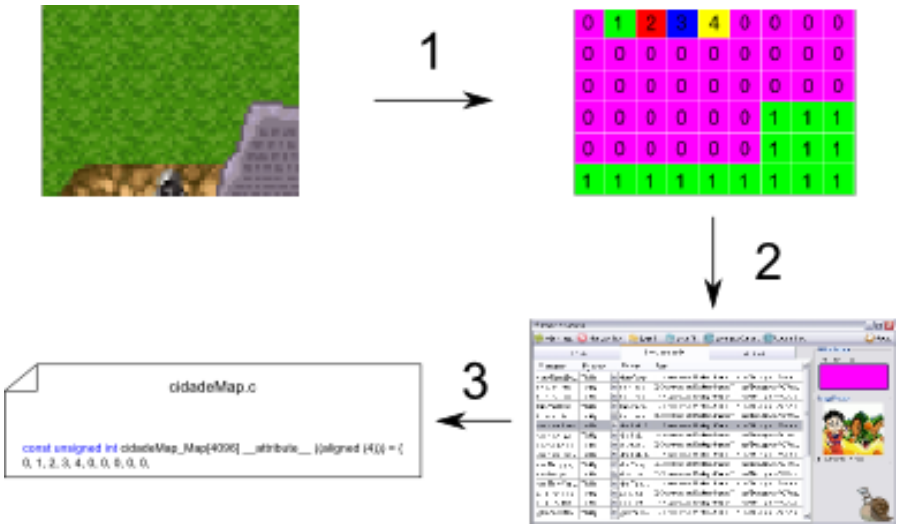

Figura 4.19: Processo de conversão de uma imagem

Para identificar em qual *tile* o personagem se encontra, é necessário transformar sua posição em *pixels* em índices de um vetor e pegar o seu valor. A partir dele é fácil decidir se o personagem está numa porta ou numa parede. A seguinte fórmula faz essa transformação:

$$indice = \left(\frac{y}{8}\right) * 64 + \frac{x}{8}$$

Para saber em qual linha da matriz o personagem se encontra, é preciso dividir a sua posição y em *pixels* por oito *pixels* (o *tile* possui dimensões 8x8) e multiplicar a quantidade de *tiles* que possui cada linha, uma vez que a imagem utilizada possui 512 *pixels* de largura. Para descobrir a coluna correspondente à posição x em *pixels*, é análoga.

A divisão $\frac{y}{8}$ possivelmente não será inteira e como o cálculo é realizado a todo instante, trabalhar com números não inteiros pode acarretar uma perda considerável no desempenho, pois o processador

gasta alguns ciclos a mais para efetuar a operação. Como o tamanho do *tile* é uma potência de dois, podemos fazer uma rotação de três *bits* para direita ($2^3 = 8$) e isso é equivalente a fazer a divisão por oito. E para efetuar essa operação, gasta-se menos ciclos do que fazer a divisão e por isso é mais eficiente. Logo, reescrevendo a fórmula acima, temos:

$$indice = (y >> 3) * 64 + (x >> 3)$$

A partir dessa expressão, é possível descobrir em qual tipo de *tile* o herói ele se encontra. Se o valor da posição *indice* do vetor for igual à zero, então ele pode andar para o *tile* seguinte. Para qualquer outro valor, o sprite do herói ficará parado ou mudará de cenário, por exemplo, se o valor do *tile* for um, o jogador ficará parado enquanto que para o valor dois, o usuário levará o herói para outro cenário.

4.12.2 Colisão entre Personagens

A colisão entre os personagens é simples. Foi determinado um raio em torno de cada um dos personagens delimitando uma circunferência como mostra a figura seguinte:

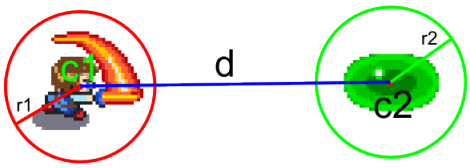

Figura 4.20: Detecção de colisão entre dois personagem

Sempre que a distância entre dois personagens for menor ou igual à soma de cada um de seus raios, então a colisão é detectada e , portanto, devem parar de andar. Logo, a seguinte fórmula resolve a colisão entre eles:

$$d = \sqrt{c1^2 + c2^2}$$

$$\text{se } d \leq (r1 + r2), \text{ para}$$

$$\text{caso contrário, não faça nada}$$

Na Batalha, essa colisão é um pouco mais elaborada devido ao raio de ataque do herói. O raio de colisão cresce na direção em que o herói está virado. Em vermelho, a circunferência de colisão do corpo do herói e em azul, a fatia que se estende quando o herói desfere seu golpe.

Figura 4.21: Esquema de colisão com espadada

4.13 Movimentando o Personagem Principal

Para movimentar o personagem principal é preciso calcular a sua posição no próximo instante a partir da posição em que ele se encontra e posição em que houve o toque na tela.

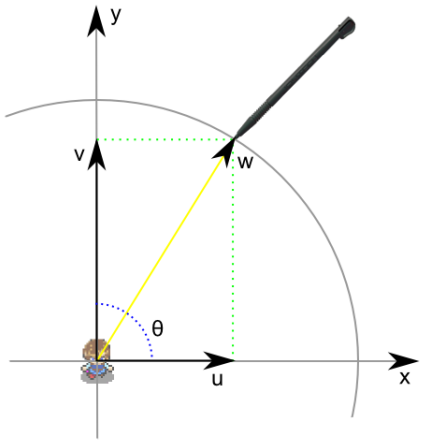

Figura 4.22: Ângulo entre dois vetores

A figura acima mostra como calcular a próxima posição do herói. O vetor w é formado pela diferença entre as posições do *stylus* e a posição do personagem. A seguinte fórmula mostra como calcular o ângulo entre os vetores w e x.

$$\theta = \arccos\left(\frac{\langle w, x\rangle}{\|w\|.\|x\|}\right)$$

A função *PA_ GetAngle()* faz exatamente isso e a partir dela obtém-se o ângulo. Com esse dado em mãos, pode-se decidir qual será a próxima posição do herói da seguinte maneira:

$$posicaoX = posicaoX + \alpha * \cos\theta$$

$$posicaoY = posicaoY - \beta * \sin\theta$$

Os valores das variáveis α e β representam o tamanho do passo do personagem. Como o personagem iria caminhar numa velocidade constante e passos de tamanho unitário, essas variáveis foram substituidas por um.

4.14 Testes

A fim de realizar testes no Nintendo DS™, o cartucho R4 Revolution foi utilizado.

Figura 4.23: Procedimento de testes no console

Após escrever um algoritmo, o programa é compilado e testado através de um emulador, No$gba. Uma vez que o algoritmo funciona, o binário *.nds* é colocado no cartucho e depois no console. Caso não funcione, o erro é procurado manualmente.

Como foi citado anteriormente, a biblioteca não é oficial, logo não oferece suporte para fazer um debug adequado e, além disso, o emulador não virtualiza fielmente o *hardware* do *video game*. Por causa desses motivos, a solução de alguns *bugs* feitas no emulador muitas vezes não eram solucionados no DS.

4.14.1 *Bugs* não Resolvidos

Ao longo do projeto muitos *bugs* foram encontrados e muitos deles resolvidos, mas alguns não foi possível solucionar, uma vez que esses problemas poderíam ser decorrentes do fato da biblioteca estar ainda em fase experimental. As principais funcionalidades não resolvidas são:

- O arquivo de música era carregado na memória, mas não era executado;

  Quando o herói vai de um cenário para outro, é esperado que o jogo troque o arquivo de música corrente, carregue outro arquivo de música e, em seguida, faça-o entrar em execução. De fato, isso acontece perfeitamente, mas em alguns momentos foi evidenciado o descarregamento do arquivo e o carregamento de outro arquivo de música, mas a execução não era feita.

- O efeito de transição entre a tela de navegação do jogo e a tela de batalha causam alguns *lags*;

  O método de transição utiliza funções trigonométricas a fim de deslocar os pixels de forma senoidal. A ocorrência de *lags* em certos momentos parecia indicar um certo *overload* de processamento. A função foi isolada do restante do jogo para que sua execução fosse estudada, mas o problema ainda persistia.

- Carregar os sprites da cidade causavam travamento no jogo.

  Ao carregar os sprites da cidade, o jogo trava. Quando o cenÃ¡rio da cidade era isolado do resto dos outros cenários, a execução ocorria de forma perfeita, mas no momento em que todos os cenários eram interligados, os sprites da cidade não eram apresentados na tela. Sem que nenhuma outra modificação no código fosse feita, o jogo, de vez em quando, transcorria de forma normal, ou seja, os sprites da cidade eram carregados normalmente e era possível jogar como esperado.

  Em alguns *debugs* realizados, tentou-se carregar os sprites em outro cenário diferente do plano de fundo da cidade. Para a surpresa do grupo, logo na primeira tentativa o carregamento dos sprites ocorria de forma perfeita, nas outras tentativa, o problema ainda persistia de forma intermitente.

Capítulo 5

Conclusão

Dado que o projeto teve duração de apenas um ano e que o desenvolvimento de um jogo demanda muito mais mão de obra do que a disponível, a etapa atual o jogo ainda não apresenta um fim. No entanto, o projeto resultou em um protótipo jogável (demo) com um considerável número de objetivos, inimigos variados com comportamentos distintos, um vasto número de NPCs que guiam o usuário pelo jogo, oito ambientes para explorar, um modo de interação intuitivo e uma interface agradável de se ver. Acompanhamentos musicais e efeitos sonoros são efeitos à parte que são responsáveis pelo maior envolvimento do usuário com o jogo.

Capítulo 6

Referência Bibliográfica

http://nocash.emubase.de/gbatek.htm

http://www.scalingweb.com/embedded_file_system.php

http://en.wikipedia.org/wiki/Embedded_system

http://www.arm.com/products/CPUs/ARM946E-S.html

http://www.vancouver.wsu.edu/fac/peabody/game-book/Coverpage.html

http://en.wikipedia.org/wiki/Nintendo_DS_Lite

http://pt.wikipedia.org/wiki/RPG_(jogo)

http://wiibrew.org/wiki/DevkitPro

http://nodadev.wordpress.com/nds-projects/efs-library/

http://www.linux.ime.usp.br/~cef/mac499-05/monografias/roberto/

http://en.wikipedia.org/wiki/Sprite_(computer_graphics)

http://en.wikipedia.org/wiki/Homebrew_%28video_games%29

http://www.devkitpro.org/about/

http://en.wikipedia.org/wiki/Nintendo_DS_homebrew

http://dldi.drunkencoders.com/index.php?title=Main_Page

http://en.wikipedia.org/wiki/ARM_architecture

http://computer.howstuffworks.com/question716.htm

http://www.touchscreenguide.com/touchscreen/res.html

http://en.wikipedia.org/wiki/Collision_detection

WHITAKER, Harold. Timing For Animation
Editora FOCAL PRESS-USA, 1ªEdição - 2006

BLAIR, Preston. Cartoon Animation
Editora Walter Foster PUB, 1ªEdição - 1995

PRESSMAN, Roger S. Engenharia de Software
Editora McGraw-Hill, 6ªEdição - 2006

Anton, Howard. Álgebra Linear com Aplicações
Editora Bookman, 8ªEdição - 2001

BUSSAB, Wilton de O. MORETIN, Pedro O. Estatística Básica
Editora Saraiva, 5ªedição - 2005

Capítulo 7

Parte Subjetiva

7.1 Marcos Takechi Hirata

Jogos de computadores sempre foram de meu interesse e algumas vezes eu ficava imaginando como seria criar um jogo com as minhas próprias histórias.

Em 2005, um dos trabalhos de formatura exposto era sobre um jogo para o Game Boy™Advance, o Nukenin, e fiquei impressionado tanto com o trabalho quanto o que um aluno do curso de ciência da computação poderia fazer.

Depois de comprar um Nintendo DS™no início do ano de 2007, comecei a procurar sobre como desenvolver jogos para o console. Alguns meses procurando um kit de desenvolvimento gratuito, consegui achar a biblioteca devkitPro, mas ainda não estava estável e não possuía uma documentação adequada e além disso, os testes que realizei nessa versão não deram muito certo.

Um ano mais tarde, encontrei uma biblioteca chamada PALib que utilizava algumas ferramentas do devkitpro e tinha uma documentação razoável. Além disso, nesse ano, a primeira biblioteca encontrada já possuía uma versão mais estável.

Para verificar se a biblioteca atendia as minhas expectativas, realizei alguns testes tanto no emulador quanto no console. Esses testes foram satisfatórios e comecei a cogitar com um amigo, no caso Nobuyuki, em fazer como trabalho de formatura um jogo para o Nintendo DS™.

Finalmente no final do ano de 2008, conseguimos a permissão do responsável pela disciplina, Prof. Carlos Eduardo Ferreira, e um orientador para nos orientar, no caso, Prof. Flávio Soares Correa da Silva.

7.1.1 Desafios & Frustrações Encontradas

Acredito que o maior desafio foi gerenciar a memória RAM limitado do Nintendo DS™, pois tivemos que aperfeiçoar a maioria dos algoritmos e muitas vezes, não tinham mais como melhorar o desempenho deles. Além disso, como a biblioteca não é oficial, não tinha como fazer um debug adequado quando há um problema na execução do programa no console.

Outra complicação foi como carregar uma música ou um cenário em tempo de execução. Para poder fazer isso, foi necessário acoplar um sistema de arquivo. Sem contar, o medo de não conseguir entregar o projeto a tempo.

Gostaria de fazer um jogo completo, com várias missões, lutas com chefões e outros, mas como temos um tempo limitado, isso não foi possível.

7.1.2 Disciplinas Relevantes e Aplicação dos Conceitos

Durante o desenvolvimento do Broken Soul, aplicamos vários conceitos estudados durante o curso. Como o projeto foi dividido basicamente em duas partes, esses conceitos foram bem ressaltados. Todas as matérias foram importantes, mas destacarei apenas as mais relevantes.

- Fase 1:

Esta fase foi caracterizada pelo planejamento do projeto.

Os conceitos aprendidos em Engenharia de Software (MAC0332) foram importantes para saber quais características que o projeto iria ter e planejar o jogo de forma sistemática para nos orientar durante o desenvolvimento. E para perceber o dinamismo do jogo, criamos um grafo, e este conceito é estudado em Algoritmo em Grafos (MAC0328). Uma das consequências de usar os conceitos de MAC0332 foi entender a arquitetura do processador. E isso foi visto em Organização de Computadores (MAC0412).

- Fase 2:

A fase dois foi a parte mais prática. A base fundamental para conseguirmos programar foram os conceitos aprendidos em Introdução à Computação (MAC0110), Princípios de Desenvolvimento de Algoritmos (MAC0122) e Estrutura de Dados (MAC0323).

Alguns problemas para serem solucionadas, foram necessários uma modelagem matemática e física, por isso que as matérias como Álgebra Linear (MAT0139) e Física 1 (FAP0126) foram importantes.

A disciplina como Introdução a Probabilidade e Estatística (MAE0121) foi necessário para implementar o reconhecimento do círculo, pois o golpe giratório dependia do tempo gasto.

Como estávamos preocupados em gerenciar a memória e o processamento, os conceitos aprendidos em Análise de Algoritmos (MAC0338) foram importantes para verificar se os nossos algoritmos tinham um tempo razoável de execução. Além disso, estávamos preocupados na interface do jogo, ou seja, se os golpes eram fáceis de fazer. Esse era exatamente um dos conceitos estudado em Princípios de Interação Homem-Computador (MAC0446).

Não posso de deixar de citar as matérias como Laboratório de Programação 1 e 2 (MAC0211 e MAC0242, respectivamente) e Laboratório de Programação Extrema (MAC0342). As duas primeiras fora importantes, pois já tínhamos uma noção de como gerenciar um projeto e a última, para agilizar o desenvolvimento.

7.1.3 Continuação na Área

Eu gosto de programar, fazer sistemas, modelagem gráfica e outros, mas em particular, eu gosto de fazer jogos e jogar é claro. Se for possível, eu pretendo continuar nessa área e futuramente, pretendo estudar algo relacionado a jogos.

7.2 Napoleão Nobuyuki Tateoka

Jogos de video-game/computadores são um grande atrativo para muita gente e eu incluo-me nesse grupo. Em 2006, navegando pelos sites de TCCs anteriores apenas por curiosidade, deparei-me com um jogo feito para o Game Boy Advance, uma outra plataforma portátil da Nintendo^{TM}anterior ao Nintendo DS^{TM}. Impressionado com o que um aluno de Ciência da Computação pode fazer, comecei a imaginar se eu poderia realizar um trabalho como aquele. Em 2008, depois de comprar um Nintendo DS^{TM}e o cartucho R4, percebi que havia uma possibilidade de existir algum kit de desenvolvimento de jogos para o console recém adquirido. Sabendo que o desenvolvimento de jogos é uma tarefa que exige muita dedicação, cogitei com mais um colega, no caso Marcos, para fazermos desse projeto um trabalho de conclusão de curso. Posteriormente, no final de 2008, a Nícia juntou-se ao grupo.

7.2.1 Desafios e Frustrações

Um dos maiores desafios para mim, assim como o Marcos citou, foi gerenciar a memória limitada do console de forma efetiva e que os algoritmos realizassem suas tarefas de modo que a animação não fosse prejudicada. No decorrer do projeto, outras dificuldades foram surgindo, tal como a necessidade de implantar um sistema de arquivos para podermos utilizar mapas gráficos mais atraentes, músicas e efeitos sonoros a fim de deixar o jogo mais interessante do ponto de vista do usuário. A criação gráfica foi outro problema. Como a utilização de imagens ou sprites já existentes acarretariam em uso ilegal, foi decidido que o próprio grupo criaria todos os efeitos gráficos do jogo. Nesse caso, fui o encarregado de criar desenhos e gerenciar o projeto de forma a torná-lo atraente. Desenhos e animações são um mundo a parte e isso foi o responsável por um grande gasto de tempo no projeto. O auxílio do grupo inteiro foi importante no acabamento dos desenhos para aumentar agilidade no trabalho. Uma outra dificuldade foi a documentação pobre fornecida pela biblioteca. Muitas vezes fomos obrigados a recorrer a fóruns de desenvolvedores mais experientes no uso da biblioteca para tentar ajudar-nos com alguns erros que, muitas vezes, nem os próprios criadores da biblioteca conseguiram responder.

A principal frustração com relaçao ao projeto é de não poder ter feito um jogo completo. Como um fã de video-game, sempre quis fazer um jogo com minhas próprias missões utilizando uma história criada por mim e, além disso, criar golpes e magias para o herói a ser controlado. Infelizmente o tempo não foi o suficiente (também por motivos de problemas de saúde) e o trabalho de programação (que já foi grande) também teria que ter sido maior, apesar de termos começado a implementação em dezembro de 2008.

7.2.2 Disciplinas Relevantes e Aplicação dos Conceitos

- MAC0110 - Introdução à Computação

- MAC0122 - Princípios de Desenvolvimento de Algoritmos

São as matérias básicas de qualquer projeto. A introdução á computação ensina os fundamentos e a lapidação do programador acontece na disciplina MAC0122. Lembro-me de um exercício-programa que pedia a análise de tempo dos algoritmos de ordenação. A discrepância dos tempos de ordenação de cada algoritmo indica-nos a importância de estudar e analisar os algoritmos cuidadosamente.

- MAC0323 - Estrutura de Dados

A escolha de uma boa estrutura define o bom funcionamento, manuseio e a facilidade de leitura do código, o que causa um impacto direto nas implementações posteriores.

- MAC0211 - Laboratório de Programação I

- MAC0242 - Laboratório de Programação II

Essas duas disciplinas me deram uma boa noção do que é trabalhar em um projeto grande. Elas incluem atividades como gerência de tarefas, modularização do código, trabalho em grupo, utilização de makefile, uso de ferramentas de controle de versões e documentação com a ferramenta .

- MAT0139 - Álgebra Linear para Computação

- MAP2210 - Aplicações de álgebra Linear

- FAP0126 - Física I

Modelagens matemáticas foram constantes no projeto. Conceitos como normas, produto vetorial, modelagens da física de projéteis, entre outros conceitos que foram constamente utilizadas/implementadas para que os requisitos do projeto fossem cumpridos. Não cursei MAT0139, visto que ainda cursava Matemática Aplicada Computacional, mas cursei duas outras matérias que juntas são equivalentes.

- MAE0121 - Introdução a Probabilidade e a Estatística I

Essa disciplina foi muito útil para guiar nos na programação do reconhecimento do círculo, tal como descrito na seção .

- MAC0420 - Introdução à Computação Gráfica

Para a criação dos gráficos, foi necessário fazer a transformação das imagens em informações mapeadas em matrizes de tranformação que farão o deslocamento dos sprites pela tela do jogo. Conceitos aprendidos na matéria também foram responsáveis pela facilitação na manipulação de programas de edição de imagens.

- MAC0414 - Linguagens Formais e Autômatos

Foi aconselhado pelo nosso orientador realizarmos um planejamento criando máquina de estados para nos orientarmos durante a programação do projeto. Desse modo rascunhos foram sendo feitos a medida que o projeto evoluia até seu estado final.

- MAC0446 - Princípios de Interação Homem-Computador

Foi uma disciplina de muita importância no desenvolvimento do projeto. Muitos conceitos aprendidos nessa matéria deram me base para a criação de uma interface limpa e intuitiva a fim de proporcionar um divertimento sem demasiadas complicações aos usuários finais.

- MAC0239 - Métodos Formais em Programação

Através dessa matéria, obtive os primeiros contatos com métodos formais para verificação da corretude dos programas, o que foi importante para o estudo e desenvolvimento dos algoritmos para a IA dos inimigos que são baseados na robustez dos códigos que executam análises matemáticas.

- MAC0332 - Engenharia de Software

A matéria de engenharia de software nos deu bases necessárias para o gerenciamento efetivo de um projeto grande com vários módulos. Através dela, aprendemos técnicas de controle de projetos que guiaram-nos de modo que o jogo pudesse ser planejado e implementado de modo sistemático e sem complicações desnecessárias.

7.2.3 Continuação na Área

Interesso me bastante pela área de desenvolvimento de jogos, área que está em ascendência no Brasil, e pretendo, se possível, continuar e trabalhar nessa área. Gostaria também de realizar estudos específios relacionados a interface gráfica e interação de usuários com com sistema.

7.3 Nícia Tiemy Sonoki

Desde que entrei no IME, em todo semestre uma das matérias que cursei passou como projeto algum tipo de jogo. Desse modo, quando o Marcos e o Napoleão falaram que iriam desenvolver um jogo para o Nintendo DS™como um trabalho de conclusão de curso, pedi a eles se poderia fazer parte do grupo. Eles disseram que sim.

Já sabendo o que iríamos fazer para o TCC, comecei a procurar trabalhos anteriores que já haviam explorado a área de jogos e achei alguns e percebi que havia muitos assuntos que podíamos abordar.

7.3.1 Desafios & Frustrações Encontradas

Um dos maiores desafios que encontrei, ao contrário dos outros integrantes do grupo, é que eu nunca havia jogado uma jogo de RPG antes. Então entendia o conceito do jogo, mas inicialmente não conseguia dar muitos *inputs* em como a dinâmica do jogo deveria ser. Aos poucos, claro, fui entendendo melhor como um jogo de *action RPG* funciona, mas ainda não possuia o *know-how* necessário. Essa foi uma frustração pessoal, pois sinto que não consegui contribuir tanto quanto gostaria.

Mesmo com essa dificuldade, acredito que consegui colaborar com o desenvolvimento, pois, uma vez escolhido como o jogo seria, ainda tínhamos que pensar em como seria o desenvolvimento.

Já na parte do desenvolvimento, como foi mencionado anteriormente, encontramos muita dificuldade pelo fato das bibliotecas usadas serem não-oficiais e, portanto, possuirem uma documentação bem pobre. Além disso, fomos encontrando vários problemas ao longo do caminho, como, por exemplo, o gerenciamento de memória.

Como o resto do grupo, também fiquei frustrada em não poder entregar um jogo mais completo. Porém, com o prazo de um ano para concluir todo o trabalho, sabíamos que isso era inviável.

7.3.2 Disciplinas Relevantes e Aplicação dos Conceitos

- MAC0110 - Introdução à Computação,
  MAC0122 - Princípios de Desenvolvimento de Algoritmos

Nessas matérias tive meu primeiro contato mais aprofundado em programação. Foi com essas matérias que percebi o quanto poderia aprender e aplicar, utilizar na prática, o que havia aprendido. Já em MAC0110 tivemos que fazer um EP que era um jogo de truco.

- MAC0323 - Estrutura de Dados

Assim como as duas disciplinas anteriores, a matéria de Estrutura de Dados me forneceu conceitos de programação básicos e indispensáveis que utilizo até hoje. Uma boa estrutura de dados é indispensável para qualquer bom algoritmo.

- MAC0211 - Laboratório de Programação I

Foi nessa matéria que desenvolvemos nosso primeiro grande projeto (que por um acaso era um jogo gráfico de plataforma). Com isso, pude aprender como funciona a estrutura de um jogo e como a documentação do projeto é importante para obter o resultado final. Foi também a primeira vez que desenvolvi um código extenso e percebi como a modularização é de fato importante e necessária.

- MAC0242 - Laboratório de Programação II

Nessa matéria, também foi implementado um projeto longo e extenso. E, novamente, tive que utilizar técnicas para gerenciar um projeto de um semestre utilizando Perl (linguagem que para mim, na época, era totalmente desconhecida).

- MAC0332 - Engenharia de Software

Essa foi uma das matérias mais importantes do curso, pois me mostrou boas práticas de programação e gerenciamento de projetos. Como o TCC é o projeto mais longo e complexo que fizemos durante o curso, saber tais técnicas foi fundamental para obter o resultado final.

- MAC0420 - Introdução à Computação Gráfica

Em Computação Gráfica, aprendemos vários conceito que foram fundamentais para entendermos as ferramentas utilizadas durante o projeto. No entanto, não foi necessário aplicá-las diretamente.

- FAP0126 - Física I

Essa matéria foi importante, pois utilizamos muitos conceitos de modelagem física e macânica para fazer os personagens se comportarem de uma maneira mais real na hora de andar, atacar e colidir.

- MAE0121 - Introdução a Probabilidade e a Estatística I

No início do projeto, não achei que utilizaria esta matéria, mas no final ela foi importante pois tivémos que fazer uma análise estatística para o Reconhecimento de Circunferências. Com isso, acabamos utilizando conceitos básicos de estatística.

7.3.3 Continuação na Área

Como já disse anteriormente, durante os cinco anos que cursei de IME, tive muito contato com programação de jogos. Porém, não pretendo seguir nessa área profissionalmente por enquanto. Acredito que o meu interesse por jogos seja mais um *hobby* e já estou pesquisando possibilidades para a criação de jogos para celulares.